Meta-programming options for DSL implementation

Ralf Lämmel
Software Languages Team
University of Koblenz
http://www.softlang.org/

http://www.softlang.org/metalib
A Chrestomathy of DSL Implementations

Simon Schauss  Ralf Lämmel  Johannes HärTEL
   Marcel Heinz  Kevin Klein  Lukas HärTEL
Software Languages Team, University of Koblenz-Landau
      Germany

Thorsten Berger
Chalmers | University of Gothenburg
       Sweden

Simon Schauss, Ralf Lämmel, Johannes HärTEL, Marcel Heinz, Kevin Klein,
Lukas HärTEL, Thorsten Berger:
A chrestomathy of DSL implementations. SLE 2017: 103-114

Benoît Combemale, Marjan Mernik, Bernhard Rumpe:
Proceedings of the 10th ACM SIGPLAN International Conference
on Software Language Engineering, SLE 2017, Vancouver, BC,
Motivation:

DSL implementation

Many paths to Rome
How to implement an FSM Language?

```python
turnstile = fsm()
    .addState("locked")
        .addTransition("ticket", "collect", "unlocked")
        .addTransition("pass", "alarm", "exception")
    .addState("unlocked")
        .addTransition("ticket", "eject", "unlocked")
        .addTransition("pass", null, "locked")
    .addState("exception")
        .addTransition("ticket", "eject", "exception")
        .addTransition("pass", null, "exception")
        .addTransition("mute", null, "exception")
        .addTransition("release", null, "locked");
```

```python
initial state locked {
    ticket/collect -> unlocked;
    pass/alarm -> exception;
}
state unlocked {
    ticket/eject;
    pass -> locked;
}
state exception { ... }
```

```
fsm {
    initial state stateA {
        eventI/actionI -> stateB;
    }
    <cell
        <no initial> state stateB {
            eventII/actionII -> stateA;
        }
    }
```
2.4 Textual syntax 15

2.4.3 Parsing text to objects

A proper parser is needed for the textual syntax once input should be processed any further. For instance, if Acme developers would want to interpret (simulate) FSMs, as in the case of the internal DSL implementation, then the parser should construct suitable objects.

Generally, a parser is supposed to map text to 'models' that can then be processed within a programming language. The term model is used here in a broad sense: instances of an object model in OO programming or instances of a domain model in modeling or model-driven engineering. This gives rise to the notion of text-to-model or, more specifically, text-to-object. One may also say that such a parser maps concrete (textual) syntax to abstract (model-based) syntax.

At Acme, the grammar of the ANTLR-based syntax checker was reused and, in fact, completed into a parser that performs a text-to-object mapping. To this end, the ANTLR grammar was extended by semantic actions for constructing Java objects according to an object model for FSML. A semantic action is basically some statement of the target language that is to be executed when parsing reaches the position of the action within the grammar rule.

Illustration 2.16 (An ANTLR-based parser description for FSML)

ANTLR resource languages/FSML/Java/FsmlToObjects.g4

```
graham FsmlToObjects;
@header {package org.softlang.fsml;}
@members {public Fsm fsm = new Fsm();}

fsm : state+ EOF ;
state :
  {boolean initial = false;}
 )
'state' stated
  fsm.getStates().add(new State($stateid.text, initial));
};
transition :
  transition* }

N.B.: AST construction behavior is modeled as embedded Java code. These so-called semantic actions are enclosed in { ... }.

N.B.: We refer to the constructed objects as ASTs (abstract syntax trees), as they abstract somewhat from the textual syntax. We populate a conservative object model because we do not care about 'fluency' in external DSL style.
A proper parser is needed for the textual syntax once input should be processed any further. For instance, if Acme developers would want to interpret (simulate) FSMs, as in the case of the internal DSL implementation, then the parser should construct suitable objects. Generally, a parser is supposed to map text to 'models' that can then be processed within a programming language. The term model is used here in a broad sense: instances of an object model in OO programming or instances of a domain model in modeling or model-driven engineering. This gives rise to the notion of text-to-model or, more specifically, text-to-object. One may also say that such a parser maps concrete (textual) syntax to abstract (model-based) syntax.

At Acme, the grammar of the ANTLR-based syntax checker was reused and, in fact, completed into a parser that performs a text-to-object mapping. To this end, the ANTLR grammar was extended by semantic actions for constructing Java objects according to an object model for FSML. A semantic action is basically some statement of the target language that is to be executed when parsing reaches the position of the action within the grammar rule.

```antlr
grammar FsmlToObjects;
@header {
package org.softlang.fsml;
}
@members {
public Fsm fsm = new Fsm();
}

fsm : state+ EOF ;
state :
    {boolean initial = false;}
    ('initial' {initial = true;})?
    'state' stateid
    {fsm.getStates().add(new State($stateid.text, initial));}
    '{' transition* '}'
    ;
transition :
    {String source = fsm.getStates().get(fsm.getStates().size() - 1).getStateid();}
    {String action = null;}
    ('/' action {action = $action.text;})?
    {String target = source;}
    ('->' stateid {target = $stateid.text;})?
    {fsm.getTransitions().add(new Transition(source, $event.text, action, target));}
    '','
    ;
stateid : NAME ;
event : NAME ;
action : NAME ;
```
Wanted: visualization

N.B.: We only require rendering of FSMs as graphs here. We could also ask for a graphical editor.
Visualization based on GraphViz (DOT)

```plaintext
digraph {
    graph [nodesep=0.5,
           rankdir=LR,
           title="Sample FSM"];
    exception [shape=ellipse];
    exception -> exception [label="ticket/eject"];
    exception -> exception [label=pass];
    exception -> exception [label=mute];
    locked [shape=ellipse,
            style=filled];
    exception -> locked [label=release];
    locked -> exception [label="pass/alarm"];           
    unlocked [shape=ellipse];
    locked -> unlocked [label="ticket/collect"];        
    unlocked -> locked [label=pass];
    unlocked -> unlocked [label="ticket/eject"];        
}
```

N.B.: We need to author code which maps FSM models to DOT code as shown.
The model of a FSM is a nested dictionary with the state ids as keys at the top, with keys 'initial' and 'transitions' per state, and with the events as keys per transition. For each event, a pair consisting of an action ('true', 'false', 'exception') and the target state is maintained. The rules underlying the JSON representation for a FSM are described as follows:

```json
{
    "exception": [{
        "initial": false,
        "transitions": {
            "release": [[""], ["locked"]],
            "ticket": [["eject", "exception"]],
            "mute": [[""], ["exception"]],
            "pass": [[""], ["exception"]]
        }
    }],
    "locked": [{
        "initial": true,
        "transitions": {
            "ticket": [["collect", "unlocked"]],
            "pass": [["alarm", "exception"]]
        }
    }],
    "unlocked": [ {
        "initial": false,
        "transitions": {
            "ticket": [["eject", "unlocked"]],
            "pass": [[""], ["locked"]]
        }
    }]
}
```

N.B.: This JSON representation is obtained by plain serialization of the dictionaries used in the Python-based object model.
import pygraphviz

def draw(fsm):
    # Create graph
    graph = pygraphviz.AGraph(title="Sample FSM", directed=True, strict=False,
                               rankdir='LR', nodesep=.5)
    # Create nodes
    for fromState, [stateDeclaration] in fsm.iteritems():
        if stateDeclaration["initial"]:
            graph.add_node(n=fromState, shape='ellipse', style='filled')
        else:
            graph.add_node(n=fromState, shape='ellipse')
    # Create edges
    for fromState, [stateDeclaration] in fsm.iteritems():
        for symbol, [(action, toState)] in stateDeclaration["transitions"].iteritems():
            graph.add_edge(fromState, toState, label=symbol + ('"" if action is None
                                            else "/"'+action))
    return graph

N.B.: We end up using an ‘internal DSL’ implementation for DOT, i.e., a library. This library takes care of rendering actual DOT textual syntax.
Ok, we can parse, check, visualize, and interpret FSMs! What else is there?

- Comprehensive **IDE support** for DSL
- Proper **embedding** of DSL into host language
- **Graphical editing** for DSL (not just visualization)
- DSL-specific **optimization** (maybe not needed for FSML)
An obvious drawback of this poorman's approach is that the proper use of the object language's syntax is not checked at compile time. Syntax errors as well as issues with conformance of the query to the underlying database schema would only be found at runtime. Perhaps a less obvious consequence of such poor checking is that programs become even vulnerable to injection attacks [BDV07, BDV10]. In this section, we focus on proper syntax-aware embedding of the object language into the metalanguage. In an extended Java language such that SQL is embedded, the above example may look as follows; this code is again adopted from [BDV07, BDV10]:

```
SQL q = <| SELECT id FROM users WHERE 
  name = ${userName} AND password = ${password} |>
```

The key idea is that, within the metalanguage (here: Java), we can embed object program fragments (here: SQL) by means of an appropriate escaping or quoting mechanism (see the `<| ··· |>` brackets) and we can escape back to the metalanguage to fill in details computed in the metaprogram (see the access to Java variables such as `userName`). Thus, syntax of object and metalanguage syntax are amalgamated in a certain manner.

### 7.5.1 Basic object syntax embedding

We will discuss here an approach to concrete object syntax which combines so-called quasi-quotation and language or syntax embedding [Mai07, Tra08, WH13]. We begin with a trivial example. Consider the following Haskell code which declares a binding for a finite state machine. We use so-called quasi-quote brackets `[fsml|···|]` (or Oxford brackets) to quote an FSM with the Haskell code.

```
Illustration 7.26 (Embedding of FSML into Haskell)
```

```
Haskell module Language.FSML.QQ.Sample

turnstileFsm :: Fsm
turnstileFsm = [fsml|
  initial state locked {
    ticket / collect → unlocked;
    pass / alarm → exception;
  }
  state unlocked {
    ticket / eject;
    pass → locked;
  }
  state exception {
    ticket / eject;
    pass;
    mute;
    release → locked;
  }

|
]
```

N.B.: We use the quasi-quote (aka Oxford) brackets `[fsml|...|]` to quote FSML code within Haskell code.
Meaning of Haskell’s quasi-quotiation

```haskell
turnstileFsm :: Fsm
turnstileFsm = Fsm [  
  State True "locked" [  
    (Transition "ticket" (Just "collect") "unlocked"),  
    (Transition "pass" (Just "alarm") "exception") ],  
  State False "unlocked" [  
    (Transition "ticket" (Just "eject") "unlocked"),  
    (Transition "pass" Nothing "locked") ],  
  State False "exception" [  
    (Transition "ticket" (Just "eject") "exception"),  
    (Transition "pass" Nothing "exception"),  
    (Transition "mute" Nothing "exception"),  
    (Transition "release" Nothing "locked") ]  
]
```

N.B.: The quasi-quoted FSM is translated at compile time to AST-based representation of FSMs which is finally represented as Haskell expression, possibly subject to constraint checking along the way.

N.B.: Concrete FSML syntax is gone. We use Haskell types for FSML’s abstract syntax.
A quasi-quoter for FSML in Haskell

fsml :: QuasiQuoter
fsml = QuasiQuoter
  { quoteExp = quoteFsmlExp
  , quotePat = undefined
  , quoteType = undefined
  , quoteDec = undefined
  }

quoteFsmlExp :: String \rightarrow Q Exp
quoteFsmlExp str = do
  \x \leftarrow parseQ fsm str
  case check \x of
    [] \rightarrow dataToExpQ (const Nothing) \x
    errs \rightarrow error \$ unlines errs

N.B.: A quasi-quoter defines how to map textual syntax appearing in different syntactic contexts (expressions, patterns, ...) into Haskell syntax.

Parse text into FSM AST
Check constraints
Map FSML to Haskell
module main::rascal::org::softlang::fsml::ConcreteSyntax

extend lang::std::Layout;

start syntax Fsm = @Foldable fsm: State* states;
syntax State = @Foldable state: "initial"? "state" Id id "{" Transition* transitions "}";
syntax Transition = transition: Event event ("/" Action action)? ("->" Id id)? ";";

syntax Id = Name;
syntax Event = Name;
syntax Action = Name;
lexical Name = ([a-zA-Z][a-zA-Z0-9]* !>> [a-zA-Z0-9]);

N.B.: Grammars are embedded into the metalanguage. In fact, an abstract syntax does not need to be defined. Metaprogramming can assume concrete object syntax; see next slide.
public set[Message] checkConstraints(Fsm f) =
  checkSingleInitial(f) +
  checkDistinctIds(f) +
  checkResolvable(f) +
  checkStateDeterministic(f) +
  checkReachable(f);

private set[Message] checkSingleInitial(Fsm f) {
  set[State] initialStates = {};
  set[State] noninitialStates = {};
  set[Message] msgs = {};
  visit(f) {
    case state: (State)`initial state <Id _> { <Transition* _> }`: initialStates += state;
    case state: (State)`state <Id _> { <Transition* _> }`: noninitialStates += state;
  }
  switch(initialStates) {
    case {}: msgs = {error("no initial state defined", n@\loc) | n <- noninitialStates};
    case {X, Y, N*}: msgs = {error("multiple initial states defined", i@\loc) | i <- initialStates};
    default: el = {};
  }
  return msgs;
}
A (Smalltalk-based) approach to language embedding

Chapter 3 Enabling Language Embedding

Helvetia provides a macro system for a high-level programming language that goes beyond changing syntax and semantics only, but that also enables language designers to adapt tools. Helvetia uses the reflective facilities of the host language and parse-tree pattern matching to identify the elements of a transformation. Transformations can be specified declaratively or imperatively. Either way, transformations preserve the mapping from the original source code to the executable code representation. This enables debuggers to know the current execution point in the untransformed source code. Thus, the source code, the abstract code representation and the executable code are causally connected. Furthermore, there is no added interpretative overhead as all behavior is compiled down to the native code representation of the host language.

3.1.1 Homogeneous Language Integration

Helvetia integrates multiple embedded languages with existing tools by leveraging and intercepting the existing toolchain and the underlying representation of the host language. Helvetia provides hooks to intercept parsing, AST transformation and semantic analysis of the standard compilation toolchain.

Figure: The code compilation pipeline of Helvetia showing multiple interception paths; there are hooks to intercept parsing <parse>, AST transformation <transform>, and semantic analysis <attribute>. Source: [RGN10]. © 2010 Springer.
Paths to advanced DSL implementation

- **Graphical editors** (e.g., GMF or Sirius)
- **Extensible languages** (e.g., Haskell with quasi-quotation)
- **Extensible compilers** (e.g., JastAddJ)
  - i.e.: language implementations that are extensible
- **Metaprogramming systems** (e.g., TXL or Stratego X/T)
  - i.e.: systems tailored towards metaprogramming
- **Language workbenches** (e.g., Xtext or JetBrains MPS)
  - i.e.: IDE for DSL development including IDE support
Conclusion?

DSL implementation

Too many paths to Rome?
MetaLib -- A Chrestomathy of DSL Implementations

Summary
MetaLib is a chrestomathy for metaprogramming. That is, the implementation of a domain-specific language is approached with different metaprogramming technologies in a manner that caters for comparison and is meant to be useful for technology documentation and learning.

Links
- The MetaLib repository: [GitHub](http://softlang.uni-koblenz.de/metalib/)
- Web-based exploration of MetaLib: [HTML](http://softlang.uni-koblenz.de/metalib/)
- Paper on MetaLib: [HTML](http://softlang.uni-koblenz.de/metalib/)
MetaLib — a chrestomathy for learning and teaching

Internal DSL style with Java with a fluent API

Snippet

Aggregated annotations

Snippet annotations

Snippet

101wiki page

© 2012-2022 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz. All rights reserved.
‘What’

What are the subjects of MetaLib?

We present … a software chrestomathy … for implementations of a domain-specific language (DSL).
## Contributions

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMFSirius</td>
<td>EMF with Sirius</td>
</tr>
<tr>
<td>EMFXMI</td>
<td>EMF with XMI persistence</td>
</tr>
<tr>
<td>EMFXtext</td>
<td>XText with derived EMF model</td>
</tr>
<tr>
<td>HaskellQuasiQuotation</td>
<td>Use of quasi-quotiation and Template Haskell</td>
</tr>
<tr>
<td>javaExternal</td>
<td>External DSL style with ANTLR and Java</td>
</tr>
<tr>
<td>javaFluentInternal</td>
<td>Internal DSL style with Java with a fluent API</td>
</tr>
<tr>
<td>javaInfluentInternal</td>
<td>Internal DSL style with Java and an influent API</td>
</tr>
<tr>
<td>MPS</td>
<td>MPS implementation based on projectional editing</td>
</tr>
<tr>
<td>pythonExternal</td>
<td>External DSL style with ANTLR and Python</td>
</tr>
<tr>
<td>pythonInternal</td>
<td>Internal DSL style with Python</td>
</tr>
</tbody>
</table>
What’s a software chrestomathy?

chrestomathy
/kərˈstəʊməθi/ ˌchər-

noun formal

a selection of passages from an author or authors, designed to help in learning a language.
Another example of a software chrestomathy

http://rosettacode.org/wiki/Rosetta_Code

Rosetta Code

Rosetta Code is a programming chrestomathy site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another. Rosetta Code currently has 850 tasks, 198 draft tasks, and is aware of 651 languages, though we do not (and cannot) have solutions to every task in every language.
Yet another example of a software chrestomathy

https://101wiki.softlang.org/

A HRMS (an information system).
Implemented in diverse languages, technologies, designs.
Characteristics of a software chrestomathy

- Community effort (for aggregation and evaluation)
- Multiplicity of languages
- Infrastructural support
- Revision and access control
- Quality assurance
- Rich metadata
- Process management

• Reference specification
Reference specification for the FSM Language

A grammar for textual syntax

```plaintext
fsm : {state}* ;
state : { 'initial'}? 'state' stateid '{' {transition}* '}' ;
transition : event { '/' action}? { '->' stateid}? ';' ;
stateid : name ;
event : name ;
action : name ;
```
Reference specification for the FSM Language

A metamodel for abstract syntax

class fsm {
    part states : state* ;
}
class state {
    value initial : boolean ;
    value stateid : string ;
    part transitions : transition* ;
}
class transition {
    value event : string ;
    value action : string? ;
    reference target : state ;
}
Reference specification for the FSM Language

Small-step operational semantics

\[
\langle \ldots, \langle b, x, \ldots, \langle e, \langle a \rangle, x' \rangle, \ldots \rangle, \ldots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle a \rangle \rangle \tag{action}
\]

\[
\langle \ldots, \langle b, x, \ldots, \langle e, \langle \rangle, x' \rangle, \ldots \rangle, \ldots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle \rangle \rangle \tag{no-action}
\]
Reference specification for the FSM Language

Negative well-formedness test case

```plaintext
initial state stateA { eventI/actionI -> stateB; }
state stateB { }
state stateC { }
```
Reference specification for the FSM Language

Generated C code

```c
enum State {LOCKED, UNLOCKED, EXCEPTION, UNDEFINED};
enum State initial = LOCKED;
enum Event {TICKET, RELEASE, MUTE, PASS};
void alarm() {}
void eject() {}
void collect() {}
enum State next(enum State s, enum Event e) {
    switch(s) {
    case LOCKED:
        switch(e) {
        case TICKET: collect(); return UNLOCKED;
        case PASS: alarm(); return EXCEPTION;
        default: return UNDEFINED;
        }
    case UNLOCKED: ...
    case EXCEPTION: ...
    default: return UNDEFINED;
    }
```

© 2012-2022 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz. All rights reserved.
‘How’

What is the MetaLib methodology?

The collected implementations are organized and documented with the help of feature modeling, semantic annotations, and model-based documentation.
Domain analysis

Legend
- Mandatory
- Optional
- Or
- Alternative
- Abstract
- Concrete

Language implementation

Syntax
Semantics

Figure 3. Feature model for metaprogramming options
Coverage of programming paradigms

We mean

As there are many implementation approaches, related technolo-

gies, and applicable languages, we decided to perform

2.2 Theoretical Sampling

We exercise

For instance, if the goal is to provide a basic, graphical editor

the concrete realizations. Hence, none of the fea-
tures to the implementation analysis phase (Sec.

2.3 Implementation Development

We developed at least one DSL implementation for each

Coverage of mainstream languages

2.4 Coverage of the Basic Feature Model

We submit the hypoth-

esis that DSLs may be implemented di-

erently depending

Coverage of DSL implementation styles

In addition to the technical documentation for exercised

5.

Legend

Mandatory

Optional

Or

Alternative

Abstract

Concrete
### 3.2.5 Translation Semantics

We decide to approach the feature of translation semantics ('simulated') to make a transition from a state with id `turnstile FSM:`

FSMs could be translated to C code with some dispatching manner, as we will substantiate in Section 4.

In the context of DSL implementation in a specific language, some metaprogramming approaches target translation or code generation in a less normative manner because we noted up-front that

- Overall there are constraints described informally as follows:
  
  - i) the ids of the declared states need to be distinct;
  - ii) the target state id must be declared;
  - iii) all states must be reachable from the initial state. For instance, here is an example violating the last constraint:
  
  as: `h` starting from initial state and an input `...`.

  - iv) the target state `h` of each transition must be declared;
  - v) all states must be distinct for each state's transitions.

- Currently, we need to explain the implementation overall and defend made changes. In particular, we need to explain the implementation overall and defend made changes. For that reason, we need to explain the implementation overall and defend made changes.

We specify the one-step relation.

That is, the FSM `...`

For instance, here is an example violating the last constraint:

As one potential approach to translation, we assume that

- That is, there are only two axioms: one for the case of an applicable transition without action, another one for an applicable transition with an action.

The reflexive, transitive closure requires a similar judgment:

That is, the finite state machine (FSM)

- `h` is applicable to an action

We consulted technical documentation and scholarly work,

where we observed that our internal DSL style implementation strategy was,

The reflexive, transitive closure requires a similar judgment:

That is, the finite state machine (FSM)

- `h` is applicable to an action

We considered now each of the seven leaf features of Fig. 2.

### Coverage of

- mainstream languages;
- programming paradigms;
- DSL implementation styles;
- technological spaces;
- the basic feature model.

<table>
<thead>
<tr>
<th>Chrestomathy member</th>
<th>Languages &amp; technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaInfluentInternal</td>
<td>Java</td>
</tr>
<tr>
<td>javaFluentInternal</td>
<td>Java</td>
</tr>
<tr>
<td>javaExternal</td>
<td>Java, ANTLR</td>
</tr>
<tr>
<td>pythonInternal</td>
<td>Python, Graphviz</td>
</tr>
<tr>
<td>pythonExternal</td>
<td>Python, ANTLR</td>
</tr>
<tr>
<td>haskellQuasiQuotation</td>
<td>Haskell (+TH+QQ)</td>
</tr>
<tr>
<td>scalaEmbedded</td>
<td>Scala</td>
</tr>
<tr>
<td>mps</td>
<td>MPS</td>
</tr>
<tr>
<td>spoofax</td>
<td>Spoofax</td>
</tr>
<tr>
<td>racket</td>
<td>Racket</td>
</tr>
<tr>
<td>rascal</td>
<td>Rascal</td>
</tr>
<tr>
<td>emfXMI</td>
<td>EMF</td>
</tr>
<tr>
<td>emfSirius</td>
<td>EMF, Sirius</td>
</tr>
<tr>
<td>emfXtext</td>
<td>EMF, Xtext</td>
</tr>
</tbody>
</table>
Implementation development

```
Branch: master  yas / languages / FSML / Java / org / softlang / fsml / fluent / Sample.java

ralmemel First l'Aquila commit.
0ac6af2 on 17 Mar

1 contributor

25 lines (21 sloc)  641 Bytes

// BEGIN ...  
package org.softlang.fsml.fluent;

import static org.softlang.fsml.fluent.FsmImpl.fsm;

public class Sample {

    public static final Fsm

    // END ...

turnstile = fsm(
        .addState("locked")
            .addTransition("ticket", "collect", "unlocked")
            .addTransition("pass", "alarm", "exception")
        .addState("unlocked")
            .addTransition("ticket", "eject", "unlocked")
            .addTransition("pass", null, "locked")
        .addState("exception")
            .addTransition("ticket", "eject", "exception")
            .addTransition("pass", null, "exception")
            .addTransition("mute", null, "exception")
            .addTransition("release", null, "locked");
```
We retransition. We determined that it is not uncommon that a code for illustration:

```
graph-like structure (in the sense of syntax); we show Java internal), we encountered a model that was closer to a 'se-
mentation, is the actual API and thus we started separating
of abstract syntax, especially in internal DSL-style imple-
object models also made us realize that an important aspect
the resulting API was more convenient (think of using a
availability of reference semantics because, in this manner,
chotomy, thereby suggesting corresponding subfeatures. We
```

Here is the corresponding re

```
4.1.1 Abstract Syntax

That is, the model is a cascaded map for maintaining states
in one internal DSL-style implementation (javaFluentIn-
The object-oriented implementations with their di
By exercising pure functional programming (in Haskell)
```

```
4.1.2 Textual Syntax

Let's focus on this node for now.
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Implementation analysis

The turnstile object in EMF's textual exchange format

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <states name="locked">  
    <transitions input="ticket" action="collect" target="unlocked"/>  
    <transitions input="pass" action="alarm" target="exception"/>  
  </states>  
  <states name="unlocked">  
    <transitions input="ticket" action="eject" target="unlocked"/>  
    <transitions input="pass" target="locked"/>  
  </states>  
  <states name="exception">  
    <transitions input="mute" action="" target="exception"/>  
    <transitions input="ticket" action="eject" target="exception"/>  
    <transitions input="release" target="locked"/>  
  </states>  
</fsml:Fsm>
```
Implementation analysis

```java
public interface Fsm {
    public Fsm addState(String state);
    public Fsm addTransition(String event, String action, String target);
    public String getInitial();
    public ActionStatePair makeTransition(String state, String event);
}
```
Implementation analysis

```
@generated
/
public class FSMImpl extends MinimalEObjectImpl.Container implements FSM {
    /*
     * The cached value of the '{@link #getStates() <em>States</em>}' containment reference list.
     * <!-- begin-user-doc --> <!-- end-user-doc -->
     * @see #getStates()
     * @generated
     * @ordered
     */
    protected EList<FSMState> states;
```
Implementation analysis

We re...

A Chrestomathy of DSL Implementations SLE’17, October 23–24, 2017, Vancouver, Canada

.../AST.scala

package org.softlang.fsml

import scala.collection.immutable.Seq

package object AST {

  case class Fsm(states: Seq[State])

  case class State(initial: Boolean, id: String, transitions: Seq[Transition])

  case class Transition(event: String, action: Option[String], target: Option[String])
}
Implementation analysis

```plaintext
FSMTransition:
   input=ID ("" action=ID)? ("" target=[FSMState])? "";
```
Implementation analysis

.../fluent/.../FsmImpl.java

```java
public class FsmImpl implements Fsm {
    private String initial; // the initial state
    private String current; // the "current" state
    // A cascaded map for maintaining states and transitions
    private HashMap<String, HashMap<String, ActionStatePair>> fsm =
        new HashMap<>();
}
```
### Implementation analysis

<table>
<thead>
<tr>
<th></th>
<th>emfSirius</th>
<th>emfXMI</th>
<th>emfXtext</th>
<th>haskellQuasiQuotation</th>
<th>javaExternal</th>
<th>javaFluentInternal</th>
<th>javaInfluentInternal</th>
<th>mps</th>
<th>pythonExternal</th>
<th>pythonInternal</th>
<th>racket</th>
<th>rascal</th>
<th>scalaEmbedded</th>
<th>spoofax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract syntax</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantic domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Textual syntax</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-CST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-AST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-ASG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projectional editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scanning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Implementation analysis

<table>
<thead>
<tr>
<th></th>
<th>emfSirius</th>
<th>emfXMI</th>
<th>haskellQuasiQuotation</th>
<th>javaExternal</th>
<th>javaFluentInternal</th>
<th>json</th>
<th>mps</th>
<th>pythonExternal</th>
<th>racket</th>
<th>rascal</th>
<th>scalaEmbedded</th>
<th>spoofax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract syntax</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASG</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantic domain</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model editing</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Textual syntax</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-CST</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-AST</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-ASG</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projectional editing</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scanning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstraction</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replacement</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Graphical syntax</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph rendering</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph editing</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic semantics</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interpretation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Static semantics</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Piggyback</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Translation semantics</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Staging</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2012-2022 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz. All rights reserved.
Semantic annotation

Data Implementation
Test Capture

Python Java
XML C
...

JUnit ANTLR3
Acceleo
...

Fluent API
Command
Macro
...

org/softlang/fsml/fluent/Sample.java

turnstile = fsm();
  .addState("locked")
  .addTransition("ticket", "collect", "unlocked")
  .addTransition("pass", "alarm", "exception")
  .addState("unlocked")
  .addTransition("ticket", "eject", "unlocked")
  .addTransition("pass", null, "locked")
  .addState("exception")
  .addTransition("ticket", "eject", "exception")
  .addTransition("pass", null, "exception")
  .addTransition("mute", null, "exception")
  .addTransition("release", null, "locked");

© 2012-2022 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz. All rights reserved.
Semantic annotation

Concept: Fluent API

---

An [[API]] where the combination of method calls is as readable as text written in a natural language

---

**Metadata**

* [[sameAs:https://en.wikipedia.org/wiki/Fluent_interface]]
* [[sameAs:https://www.martinfowler.com/bliki/FluentInterface.html]]
* [[relatesTo:https://dzone.com/articles/java-fluent-api-design]]
* [[isA::API]]
## Information retrieval
informing implementation analysis and semantic annotation

<table>
<thead>
<tr>
<th>Burmako13 (SCALA)</th>
<th>EfftingeV06 (XTEXT)</th>
<th>Hudak98 (HASKELL)</th>
<th>KatsV10 (SPOOFAX)</th>
<th>Mainland07 (HASKELL)</th>
<th>Parr13 (ANTLR)</th>
<th>Rascal11 (RASCAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>model</td>
<td>language</td>
<td>language</td>
<td>haskell</td>
<td>rule</td>
<td>rascal</td>
</tr>
<tr>
<td>macros</td>
<td>used</td>
<td>can</td>
<td>used</td>
<td>language</td>
<td>token</td>
<td>value</td>
</tr>
<tr>
<td>scala</td>
<td>text</td>
<td>programming</td>
<td>rule</td>
<td>quasiquoting</td>
<td>parser</td>
<td>page</td>
</tr>
<tr>
<td>compiler</td>
<td>generated</td>
<td>used</td>
<td>editor</td>
<td>type</td>
<td>grammar</td>
<td>int</td>
</tr>
<tr>
<td>language</td>
<td>code</td>
<td>interpreter</td>
<td>can</td>
<td>syntax</td>
<td>antlr</td>
<td>set</td>
</tr>
<tr>
<td>generated</td>
<td>can</td>
<td>function</td>
<td>type</td>
<td>data</td>
<td>expr</td>
<td>type</td>
</tr>
<tr>
<td>class</td>
<td>language</td>
<td>semantic</td>
<td>services</td>
<td>used</td>
<td>java</td>
<td>list</td>
</tr>
<tr>
<td>programming</td>
<td>file</td>
<td>region</td>
<td>generated</td>
<td>exp</td>
<td>can</td>
<td>used</td>
</tr>
<tr>
<td>used</td>
<td>check</td>
<td>domain</td>
<td>spoofax</td>
<td>code</td>
<td>used</td>
<td>exp</td>
</tr>
<tr>
<td>def</td>
<td>name</td>
<td>time</td>
<td>syntax</td>
<td>generated</td>
<td>parse</td>
<td>str</td>
</tr>
<tr>
<td>implicit</td>
<td>type</td>
<td>dsl</td>
<td>development</td>
<td>programming</td>
<td>lexer</td>
<td>statement</td>
</tr>
<tr>
<td>can</td>
<td>message</td>
<td>haskell</td>
<td>specific</td>
<td>function</td>
<td>match</td>
<td>programming</td>
</tr>
</tbody>
</table>
Semantic annotation
Model-based documentation

https://github.com/softlang/metalib/blob/master/models/javaInfluentInternal.json

```json
{
    "name": "javaInfluentInternal",
    "baseuri": "https://github.com/softlang/yas/tree/master/languages/FSML/Java/org/softlang",
    "headline": "Internal DSL style with Java with a fluent API",
    "sections": [
        {
            "features": ["API"],
            "perspectives": ["data"],
            "languages": ["Java"],
            "concepts": ["Fluent API"],
            "technologies": [],
            "artifacts": [{"type": "all", "link": "fsml/fluent/Sample.java"}]
        },
        ...
    ]
}
```
Metamodel of documentation

// Documentation of contributions

class document {
    value name : string; // The name of the contribution
    value headline : string; // A one-liner explanation
    value baseuri : string; // Base URI for links
    part sections : section+; // Sections of the documentation
}

// Sections in a documentation

class section {
    value headline : string?; // Optional one-liner explanation
    part perspectives : perspective+; // Perspective of section
    value features : string+; // Features addressed by section
    value languages : string*; // Languages used
    value technologies : string*; // Technologies used
    value concepts : string*; // Concepts used
    part artifacts : artifact+; // Artifacts to be shown
}

// Perspectives of documentation

// Document artifacts that are related to features in other ways.
// Perspectives of documentation
enum perspective {
    implementation, // i.e., feature implementation
data, // e.g., instance of grammar or metamodel
test, // i.e., application of implementation
build, // e.g., code generator application
capture // e.g., screenshot or session log
}

// Artifacts for projected by section
abstract class artifact {
    value link : string; // A relative URI
    value format : string; // MIME-like format type
}
class none extends artifact { } // Nothing to show
class all extends artifact { } // All to show
class some extends artifact { // A specific line range to show
    value from : integer;
    value to : integer;
}
‘Why’

What scenarios for learning and teaching exist?

The chrestomathy is useful for learning (teaching) in so far that it provides a high level of abstraction for metaprogramming and it directly enables the side-by-side exploration of implementation approaches for DSLs (so that one can learn new metaprogramming techniques based on techniques already known).
Which DSL implementation uses an API?

Feature: API

Contributions

- EMFSirius
- EMFXML
- javaFluentInternal
- javalnfluentInternal
- pythonExternal
- pythonInternal
- Rascal

101Wiki

https://101wiki.softlang.org/Feature:API
What is a fluent API?

Concept: Fluent API

Contributions

- javaFluentInternal
- pythonInternal
- Rascal

101Wiki

https://101wiki.softlang.org/Fluent_API
Where is the API implemented?

```java
public interface Fsm {
    public void addState(String state);
    public void addTransition(String event, String action, String target);
    public String getInitial();
    public ActionStatePair makeTransition(String state, String event);
}
```

```java
// Helper class for "makeTransition"
public class ActionStatePair {
    public String action;
    public String state;
}
```

How does influent differ from fluent Java implementation?

Internal DSL style with Java with a fluent API

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Features</th>
<th>Languages</th>
<th>Technologies</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>API</td>
<td>Java</td>
<td>JUnit</td>
<td>Fluent API</td>
</tr>
<tr>
<td>Implementation</td>
<td>Interpretation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Semantic domain</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Internal DSL style with Java and an influent API

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Features</th>
<th>Languages</th>
<th>Technologies</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>API</td>
<td>C</td>
<td>JUnit</td>
<td>Batch file</td>
</tr>
<tr>
<td>Implementation</td>
<td>AST</td>
<td>Java</td>
<td>StringTemplate</td>
<td>Functional constructor</td>
</tr>
<tr>
<td>Test</td>
<td>Compilation</td>
<td></td>
<td></td>
<td>Generalised code</td>
</tr>
<tr>
<td></td>
<td>Interpretation</td>
<td></td>
<td></td>
<td>OO class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Template processing</td>
</tr>
</tbody>
</table>
As illustrated in Fig. 6, roles ('hats'): the development role ('code' in the figure) in
5. Model-based Documentation

tations according to the theoretical sampling of Section 2.2.
4.3 Coverage of Feature Model

Staging
Translation semantics
Analysis
Static semantics
Interpretation
Text projection
Model projection
Projectional syntax
Graph editing
Graph rendering
Graphical syntax
Replacement
Abstraction
Scanning
Text-to-ASG
Text-to-CST
Textual syntax
Resolution
Serialization
Semantic domain
ASG
AST
Abstract syntax

© 2012-2022 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz. All rights reserved.
Future work

• Add **contributions**.

• Add **features**.

• Refine **theoretical sampling**.

• Advance the use of **IR techniques**.

• Define and improve quality of **101wiki**.

• **Cross-validate** contributions and documentation.

• Evaluate MetaLib in **classroom**.

Status at the end of 2022: Project stalled. May need a restart.
Meta-programming options for DSL implementation

End of slide deck

http://www.softlang.org/metalib